applied sciences

MDPI

Article

Application of a Simulation-Based Digital Twin for Predicting Distributed Manufacturing Control System Performance

Gonçalo Roque Rolo [1], Andre Dionisio Rocha [1,2,*], João Tripa [2] and Jose Barata [1,2]

[1] Department of Electrical and Computer Engineering, NOVA School of Science and Technology, NOVA University of Lisbon, 2829-516 Caparica, Portugal; ga.rolo@campus.fct.unl.pt (G.R.R.); jab@uninova.pt (J.B.)
[2] UNINOVA—Centre of Technology and Systems (CTS), FCT Campus, Monte de Caparica, 2829-516 Caparica, Portugal; j.tripa@uninova.pt
* Correspondence: andre.rocha@uninova.pt

Abstract: During the last years, several research activities and studies have presented the possibility to perform manufacturing control using distributed approaches. Although these new approaches aim to deliver more flexibility and adaptability to the shop floor, they are not being readily adopted and utilised by the manufacturers. One of the main challenges is the unpredictability of the proposed solutions and the uncertainty associated with these approaches. Hence, the proposed research aims to explore the utilisation of Digital Twins (DTs) to predict and understand the execution of these systems in runtime. The Fourth Industrial Revolution is leading to the emergence of new concepts amongst which DT stand out. Given their early stage, however, the already existing implementations are far from standardised, meaning that each practical case has to be analysed on its own and solutions are often created from scratch. Taking the aforementioned into account, the authors suggest an architecture that enables the integration between a previously designed and developed agent-based distributed control system and its DT, whose implementation is also provided in detail. Furthermore, the digital model's calibration is described jointly with the careful validation process carried out. Thanks to the latter, several conclusions and guidelines for future implementations were possible to derive as well.

Keywords: Cyber-Physical Production System; Digital Twin; distributed control systems; industrial agents; Multi-Agent System; simulation

check for updates

Citation: Roque Rolo, G.; Dionisio Rocha, A.; Tripa, J.; Barata, J. Application of a Simulation-Based Digital Twin for Predicting Distributed Manufacturing Control System Performance. *Appl. Sci.* **2021**, *11*, 2202. https://doi.org/10.3390/app11052202

Academic Editor: Alexandre Carvalho

Received: 7 February 2021
Accepted: 24 February 2021
Published: 3 March 2021

1. Introduction

As a result of the German government's pioneer effort to innovate manufacturing systems [1], the Fourth Industrial Revolution was designated Industry 4.0 (I4.0) for the first time in 2011, on the occasion of the Hannover Fair [2].

Despite the term's extensive nature, a widely accepted definition can be found in [3], where it is described as "a set of technologies based on digitisation and interconnection of all production units present within an economic system".

In order to improve the efficiency and profitability of manufacturing systems [4] by integrating activities involving human beings, machines and data [2], I4.0 relies on enabling technologies such as big data, Internet of Things (IoT) and simulations. Therefore, factories are progressively becoming smart factories, the main features of which are the capacity to accept on-the-fly changes to the ongoing manufacturing processes, provide remote access to every single resource and autonomously organise production tasks [5].

Nevertheless, in spite of I4.0's indisputable benefits, the high initial investments, the uncertainty about its profitability and the risk of cyber-attacks, still prevent companies from embracing the transition [2].

Tightly related to I4.0 is the concept of Cyber-Physical System (CPS) [6], coined by Helen Gill at a workshop on CPSs, hosted by the National Science Foundation (NSF) in

2006 [7]. According to [8], "*CPSs are systems of collaborating computational entities which are in intensive connection with the surrounding physical world and its on-going processes, providing and using, at the same time, data-accessing and data-processing services available on the internet*".

Whenever applied for manufacturing purposes, CPSs are referred to as Cyber-Physical Production Systems (CPPSs) [9]. Most of the proposed CPPSs aim to bring dynamic and flexible solutions using adaptable and reconfigurable distributed control systems [10,11]. Usually, these distributed control systems are modelled as modular components and designed using Multi-Agent Systems (MASs) approaches [12]. Although the benefits and improvements introduced by these solutions are known and understandable by the manufacturers, it is still difficult for these approaches to be adopted and used in real industrial environments. In the literature, some authors already identified barriers as critical in adopting these approaches [13]. One of the main obstacles is the difficulty to predict the self-organised behaviour of such systems and the impossibility to understand how the system will evolve [14]. Hence, this research aims to study the utilisation of Digital Twins (DTs) to mitigate this aspect. The authors believe that a simulation-based DT can be used to predict the system's behaviour and performance. To do that, it must use the production line's current status and the production plan in order to simulate the system to different time horizons.

The main innovation associated with the previous concepts is related to the digital world and its components. On this regard, an identical concept to what is now referred to as DT was originally suggested by Michael Grieves on the occasion of an industry lecture on Product Lifecycle Management (PLM). According to him, a DT was no more than a virtual entity capable of mirroring its physical counterpart [15]. It is worth mentioning that, in spite of this definition still being suitable, a considerable amount of authors mistakenly insist on stating that a DT includes the components from both the cyber and the physical worlds.

According to [16], three designations are possible depending on the integration level between the physical asset and its digital corresponding. In case the data exchange between both entities is manually performed, the term Digital Model is adopted. If only the physical device is able to automatically update its counterpart, the term Digital Shadow is used instead. The full DT level implies that the cyber entity is capable of influencing its physical twin and vice versa.

In any case, Key Performance Indicators (KPIs) may be used to assess the models' accuracy. These result from the combination of Key Result Indicators (KRIs), which are directly measurable, and Performance Indicators (PIs), that are mathematically obtainable using the latter [17].

Therefore, DTs have proven to be powerful tools when it comes to ensuring reasonable proximity between the established goals and the real outcome, enhancing production processes and quality check, as well as monitoring and making predictions related to the manufacturing systems [18].

According to [16], more than half of the analysed articles approached DTs in a conceptual way. Regarding the level of integration, 35% only achieved the Digital Shadow level, 28% dealt with Digital Models and no more than 18% were considered full Digital Twins.

In [5], a semantic model was developed with the aim of tackling resource virtualisation. For that, Ontology Web Language (OWL) was used as the ontology language and Jena as the rule language for resource capability description. Describing a machine's attributes and abilities through semantic web languages allows the representation of real-world information in a comprehensive format for computing. Furthermore, ontologies promote the models' scalability through their objects' properties [19].

The practical application described in [20] consists of a high-fidelity model of a physical object focusing on its geometry. For that, ISO 10303, or simply STEP, was used. Besides the precise representation of geometrical objects through standardised terms, STEP excels at the *.stp* files' compatibility. Nevertheless, this application only represents the physical shape of an object, disregarding its behaviour.

In [21], a DT was created to monitor the energy consumption of a I4.0 laboratory. The Open Platform Communications Unified Architecture (OPC-UA) was used for communication amongst machines and data acquisition. Aside from being a step towards standardisation, this protocol has an embedded security layer, ruling out the need for further safety rules. In addition to Simulink, both MATLAB and its OPC Toolbox were vital for integrating data exchange between the servers and the simulation, given that all the Programmable Logic Controllers (PLCs) were compatible with this communication protocol.

At last, in [22], the DT concept was applied to a rear lights' production system. In order to avoid the need for integration, two models were created using AnyLogic: the front-end, representing the physical system which only executed tasks, and the back-end, representing the actual DT which was responsible for giving orders to the former. After connecting both models using a Java interface, several simulations were carried out and the following conclusions were drawn:

- Throughput time was reduced by lower failure rates and available feedback from each station and higher amounts of sensors along the railway.
- The amount of lost messages increases with the number of sensors due to concurrent communications.
- In spite of delaying communications, a higher number of sensors reduces throughput time, in particular when stations' failure rate increases.

Overall, implementing DTs raises a number of issues related to communication speed constraints, the data acquisition method, the lack of standards that ensure scalability of existing solutions and unawareness of how to integrate human beings in the process. Therefore, most applications only support workers in the decision-making process [23].

2. Simulation-Based Digital Twin Framework

2.1. Architecture

With the aim of covering the aforementioned gaps, a three-layered architecture for a DT of a previously developed distributed control system was proposed, as portrayed by Figure 1.

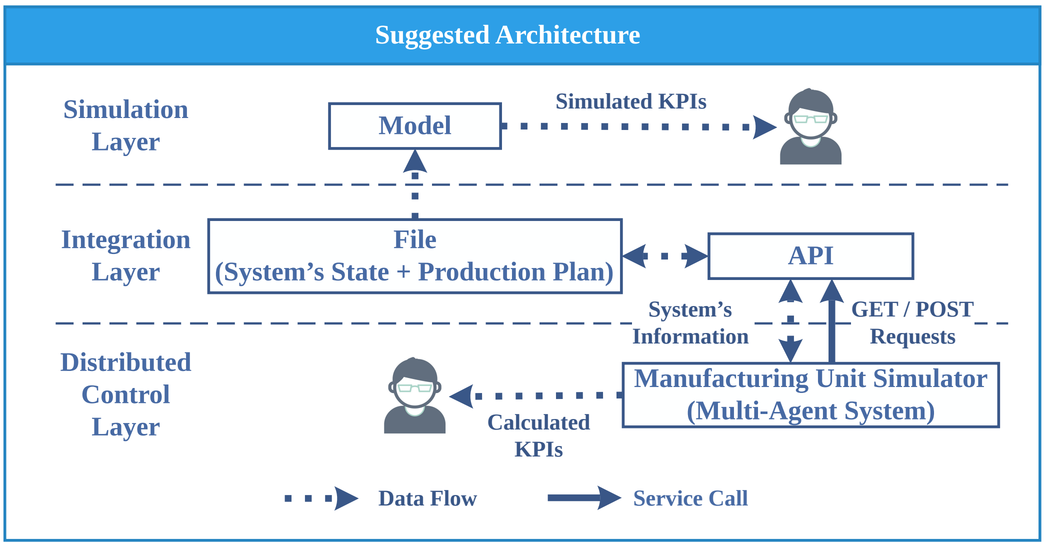


Figure 1. Suggested three-layered architecture.

The Distributed Control Layer (DCL) contains the demonstrator from [24], controlled by a fully working MAS capable of dynamically adapting its behaviour according to the production requirements and the available stations.

The data model designed in [24] in order to abstract this MAS is presented in Figure 2. According to Figure 2, this data model describes four types of entities:

- Conveyor—each conveyor, from conveyorA (entry) to conveyorF (exit), is able to carry one product at a time.
- Resource—each resource is capable of adding/removing a product to/from the line or brushing it. A resource must be associated with one of the conveyors.
- Skill—corresponds to each type of action performed by a resource.
- Product—each product consumes different amounts and types of skills, i.e., products from type one consume one BrushUp, products from type two consume one BrushDown and products from type three consume one BrushUp and one BrushDown. More types can be easily added by editing the ontology.

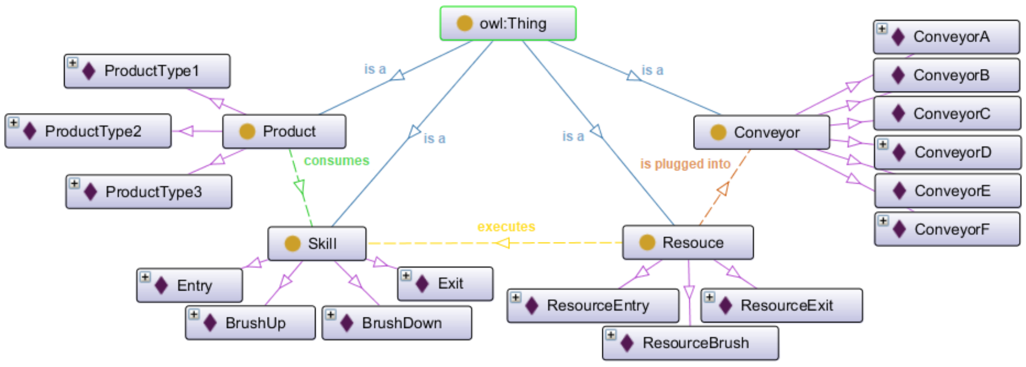


Figure 2. Multi-Agent System's data model.

When launched, the MAS remains idle until a suitable production plan is introduced, by which time the first product is added to the conveyor network. The deployment of the remaining products happens periodically, after the entry conveyor's availability is ascertained. Whenever a product is removed from the line, the KPIs are updated. These events occur as many times as needed until the whole production plan has been processed.

The Integration Layer (IL) is responsible for making the physical system's relevant information available to external applications. It contains a file, that plays the role of a database, where the demonstrator's current state and the remaining production plan are stored, and an Application Programming Interface (API), whose purpose is to interact with the database. For that, the latter provides two services which endure concurrent access. By sending a *GET* request with no specific message, the API is instructed to return the production plan, after reading the file. On the other hand, through a *POST* request, whose message body must contain the system's current state and the remaining production plan, one may ask the API to update the file. The first operation comes in handy before the insertion of a product, as it allows the client to check the next product's type, whereas the second is used whenever a product is added to the production line, changes position, its processing ends or when the production plan is first created.

Finally, the Simulation Layer (SL) contains the digital model. So as to serve its purpose, a correct description of the system's initial state and the remaining production plan are required. This way, the model knows where to start simulating from and the type of products to be inserted afterwards. Additionally, the user is expected to configure the time interval between the insertion of two consecutive products and the delay associated with the consumption of each type of skill, according to the data model. Similarly to its physical twin, the digital model updates the KPIs whenever a product leaves the system, even though the simulation only ends upon the production plan's completion.

The interaction between all these modules is described in the cross-functional diagram from Figure 3.

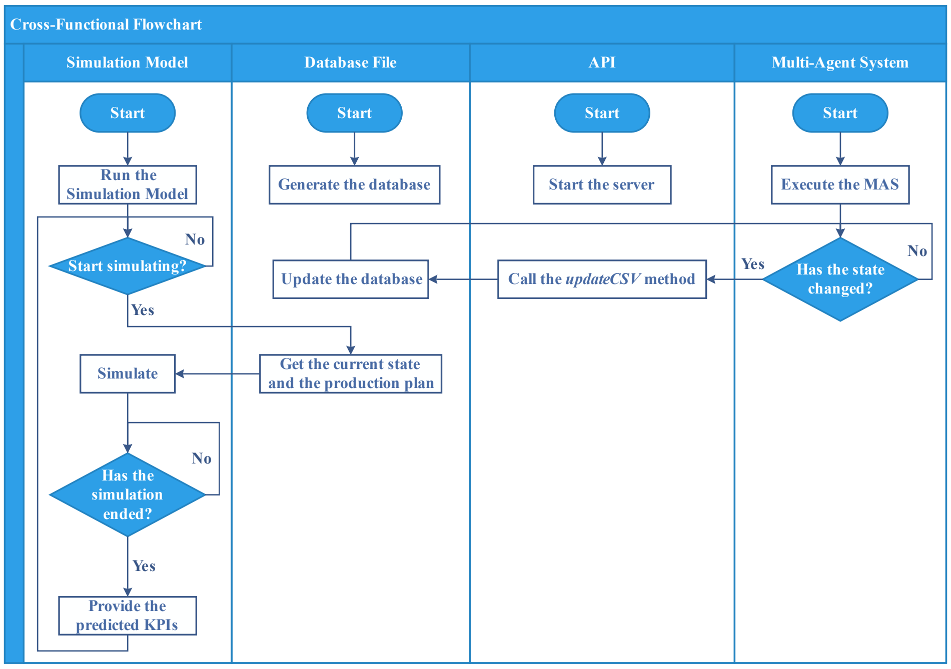


Figure 3. Cross-functional flowchart.

Although the authors could use any KPIs to develop and demonstrate the proposed approach's utilisation and performance, some constraints needed to be considered due to the system modelled. Since the demonstrator used for the research does not effectively produce or assemble a product, it is impossible to measure waste or quality values. Hence, only time-related KPIs could be used to evaluate the proposed research. In the literature, it is possible to find many contributions presenting KPIs commonly used to evaluate manufacturing systems' performance. For the proposed work, the authors selected cycle time [25] and throughput [26] to guide the developments.

2.2. *Implementation*

The subsections below intend to clarify the implementation of each segment from the architecture, following top-down approach of the architecture.

2.2.1. The Simulation Model

The digital model itself was developed using AnyLogic, a powerful and well-documented agent-based simulation tool.

Firstly, two types of agents were created:

- *Main*—includes all graphical and logical elements, responsible for ascertaining the model's correct behaviour until the final predictions are derived.
- *Product*—abstracts a product and describes it through four parameters: *brushUp* and *brushDown*, that specify the type, and *entryTime* and *exitTime*, where the respective timestamps are stored. Unlike the *Main* type, there can be more than one instance per simulation.

Afterwards, the physical system was represented using Conveyors, Positions on Conveyors and Transfer Tables, available at the Space Markup category from the Material Handling Library. The result is shown in Figure 4.

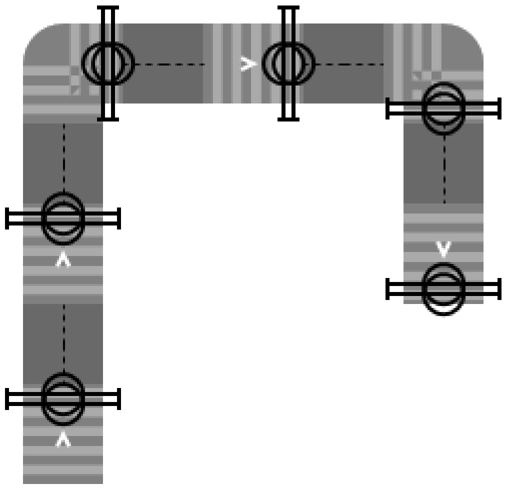


Figure 4. Conveyor network's 2D view.

With the aim of controlling the model's overall behaviour, the blocks diagram represented in Figure 5 was designed.

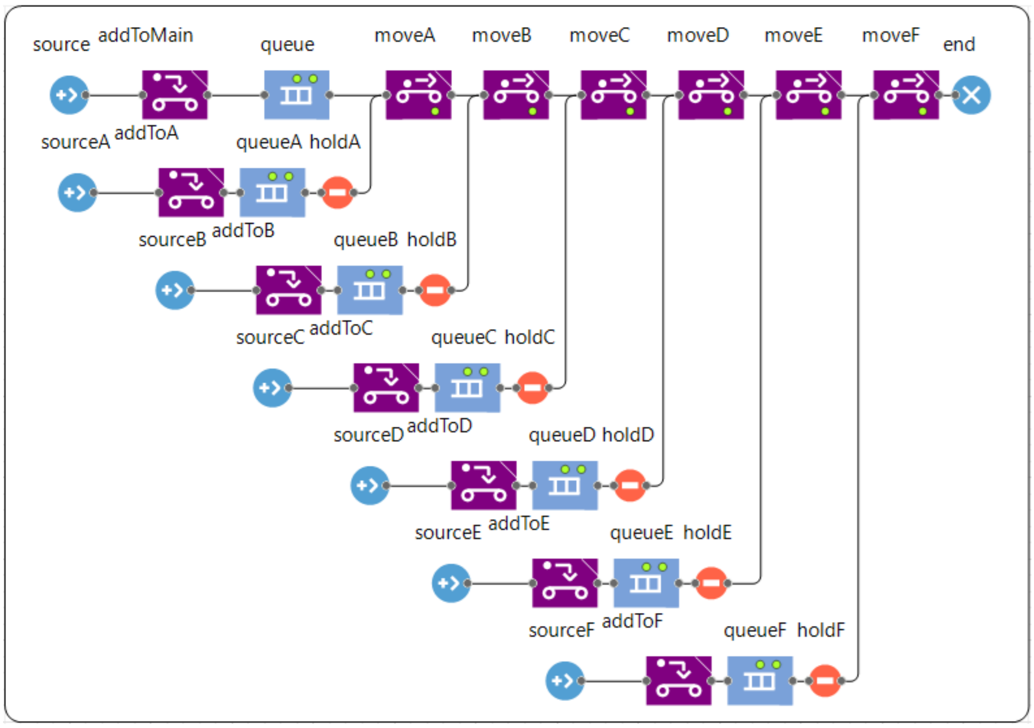


Figure 5. Simulation model's blocks diagram.

For that, the following elements from the Blocks category of the Material Handling Library were used:

- Source—launches a new *Product* instance. The blocks from *sourceA* to *sourceF* deploy a product on the respective conveyor after the initial state is read from the file, whereas the *source* block launches the remaining products.
- Conveyor Enter—is placed after a source block and establishes, through Position on Conveyor elements, the newly-created product's initial position.

- Queue—stacks products in case they cannot be immediately added to the conveyor network.
- Hold—blocks the natural course of an agent. It ensures that all products, from the initial state, were added before the conveyors start moving.
- Convey—settles the sequence of conveyors that constitute the network by defining each segment of the path.
- Sink—works as the endpoint of the production line, where products are dispatched.

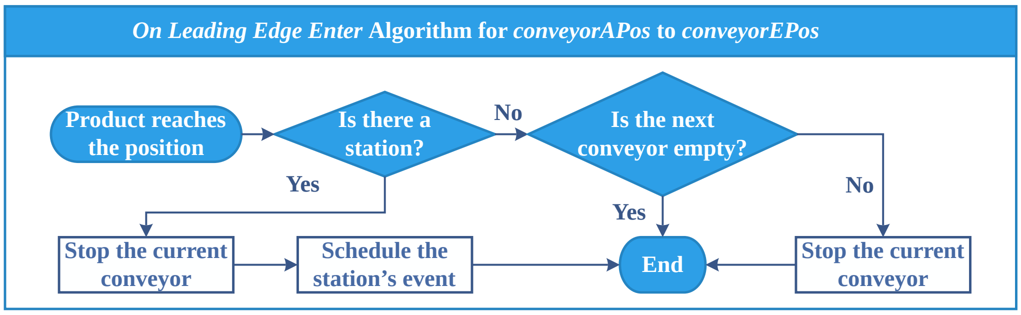
However, the diagram is incapable of limiting the amount of simultaneous products per conveyor and emulating the delays associated with the processing stations. Thus, several events (one that controls the products' insertion and one for each conveyor's station), variables (control flags, statistics data, etc.), parameters and functions had to be used jointly with some inbuilt methods, so as to model the aforementioned behavioural nuances.

One such example is the *loadState* function, the first to be invoked immediately after the simulation starts, which is responsible for placing the due products on the respective conveyors, according to the initial state described in the database file. Additionally, it computes the remaining processing time for every suitable product and accordingly schedules the corresponding station's event. Given that this method is only invoked at the beginning of the execution, the model is insensible to changes in the physical system's configuration during the simulation. For that reason, it is assumed that the stations' location remains unchanged throughout the entire execution.

The *insertProduct* method plays a similarly vital role. It is called whenever the insertion's event reaches the timeout condition, by which time a product is injected on the network, provided that the entry conveyor is empty. If otherwise, the insertion's event is called off because the next attempt is made as soon as conveyor A becomes available.

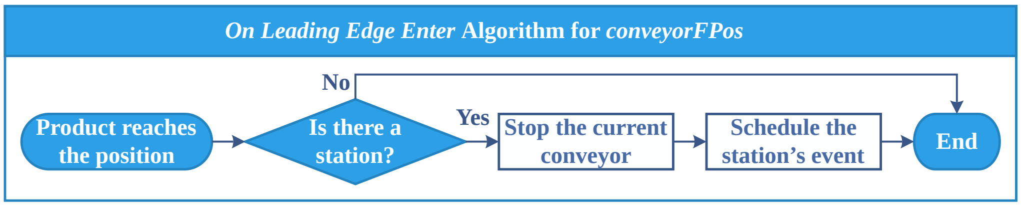
Furthermore, some occurrences throughout the simulation require extra algorithms. For instance, a product's leading edge crossing a Position on Conveyor element symbolises the arrival at a station. For that reason, each of these elements had to be endowed with the algorithms below.

According to Figure 6, when a product arrives at a conveyor with a station, the conveyor is stopped and an event that emulates the processing delay is scheduled. Even if otherwise, if the next conveyor is busy, the current one is equally stopped. As for conveyor F, Figure 7 shows that the corresponding piece of logic does not ascertain the following conveyor's availability, since it corresponds to the network's last conveyor.



On Leading Edge Enter Algorithm for *conveyorAPos* to *conveyorEPos*
Product reaches the position
Is there a station?
No
Is the next conveyor empty?
Yes
No
Yes
Stop the current conveyor
Schedule the station's event
End
Stop the current conveyor

Figure 6. On leading edge enter algorithm for *conveyorAPos* to *conveyorEPos*.



On Leading Edge Enter Algorithm for *conveyorFPos*
Product reaches the position
No
Is there a station?
Yes
Stop the current conveyor
Schedule the station's event
End

Figure 7. On leading edge enter algorithm for *conveyorFPos*.

Another example is the arrival of a product at a conveyor, which entails some operations.

According to Figure 8, when a product reaches conveyor B, if there is a product to be inserted and the last insertion was unsuccessful, another attempt is made and the event that controls the periodic insertions is restarted. Afterwards, conveyor B starts moving. If otherwise, conveyor B is simply set to run. This is the most complex version of the algorithm, due to the fact that conveyor B is placed immediately after the entry conveyor. Thus, the algorithm for conveyors C to F does not deal with products' insertion, as portrayed by Figure 9. In this case, the only taken action is running the respective conveyor. As for conveyor A, no extra logic was added to the *On leading edge enter* field because products never cross its beginning.

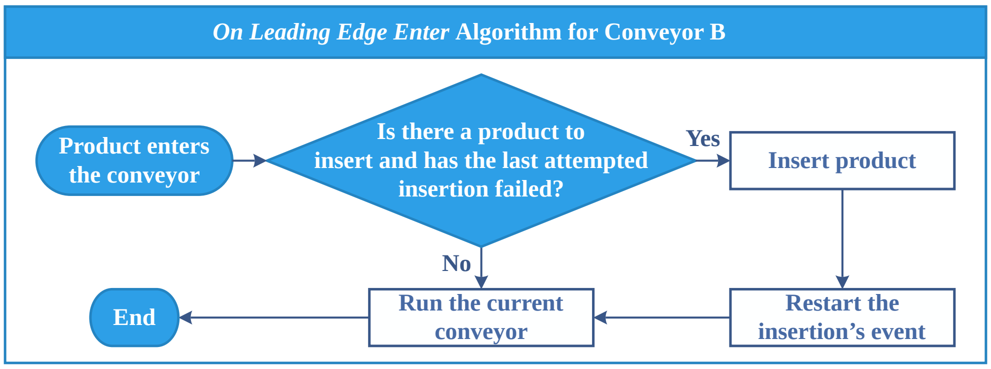

On Leading Edge Enter Algorithm for Conveyor B
Product enters the conveyor
Is there a product to insert and has the last attempted insertion failed?
Yes
Insert product
No
Restart the insertion's event
Run the current conveyor
End

Figure 8. On leading edge enter algorithm for conveyor B.

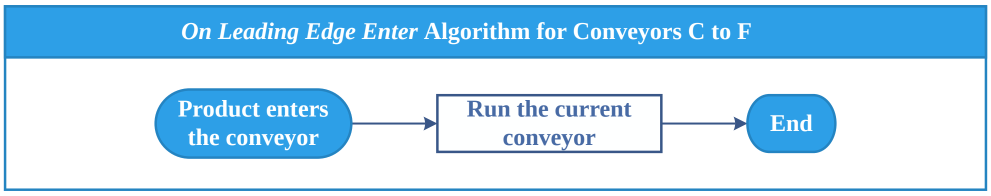

On Leading Edge Enter Algorithm for Conveyors C to F
Product enters the conveyor
Run the current conveyor
End

Figure 9. On leading edge enter algorithm for conveyors C to F.

Ultimately, a product leaving a conveyor is also a key-moment for the model's behaviour, requiring some specific actions as well.

Once a product leaves conveyor A, it is stopped, as shown in Figure 10. Additionally, for the other conveyors, if the previous conveyor has a product that needs not be processed, that conveyor starts moving, as presented in Figure 11. As for conveyor F, the *On trailing edge exit* field was left empty on the account of products being removed from the line before their trailing edge crosses the exit conveyor. Hence, the equivalent algorithm for this conveyor had to be inserted on the *On enter* field of *end* from the blocks diagram. In this case, according to Figure 12, the KPIs' predictions are updated as well.

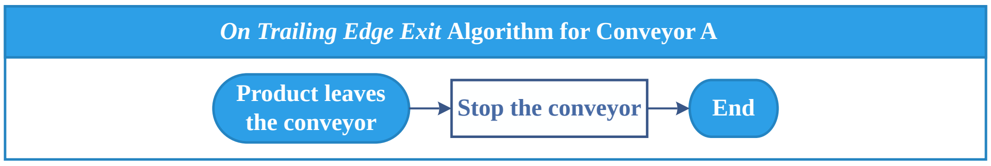

On Trailing Edge Exit Algorithm for Conveyor A
Product leaves the conveyor
Stop the conveyor
End

Figure 10. On trailing edge exit algorithm for conveyor A.

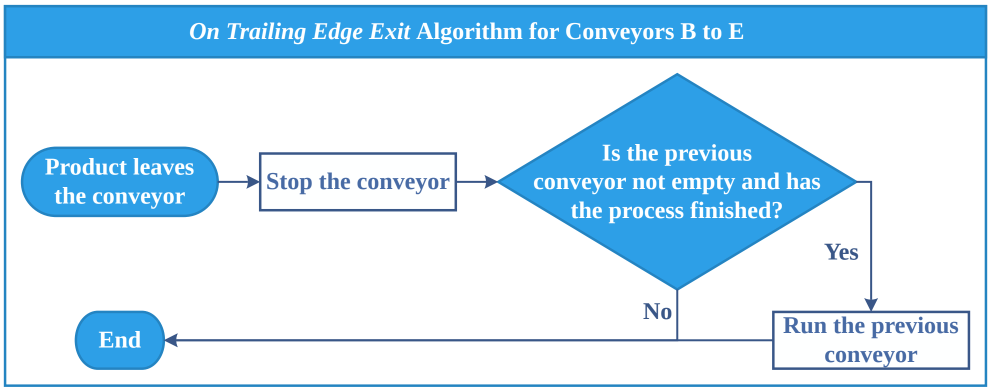

On Trailing Edge Exit Algorithm for Conveyors B to E
Product leaves the conveyor
Stop the conveyor
Is the previous conveyor not empty and has the process finished?
Yes
No
Run the previous conveyor
End

Figure 11. On trailing edge exit algorithm for conveyors B to E.

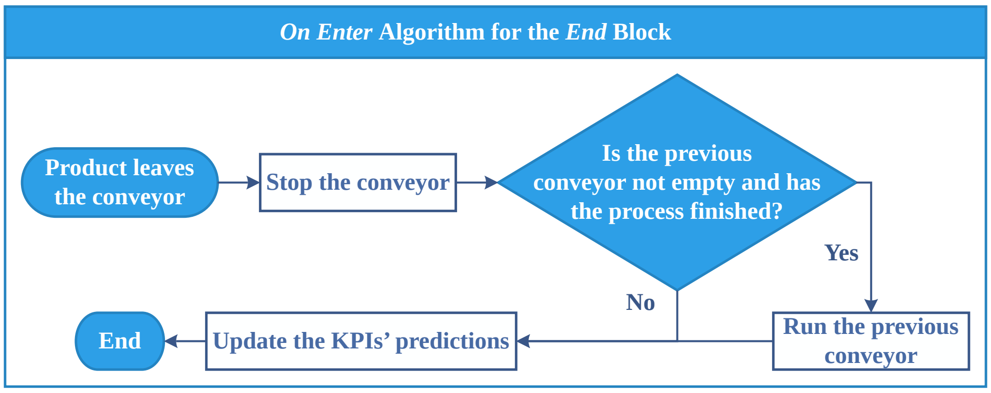

On Enter Algorithm for the *End* Block
Product leaves the conveyor
Stop the conveyor
Is the previous conveyor not empty and has the process finished?
Yes
No
Run the previous conveyor
Update the KPIs' predictions
End

Figure 12. On enter algorithm for the *End* block.

The last stage of the model's implementation consisted of designing a Graphical User Interface (GUI) with the aim of providing the end-user with an intuitive way of interacting with the simulation model. As soon as the application starts running, the homepage from Figure 13 pops up.

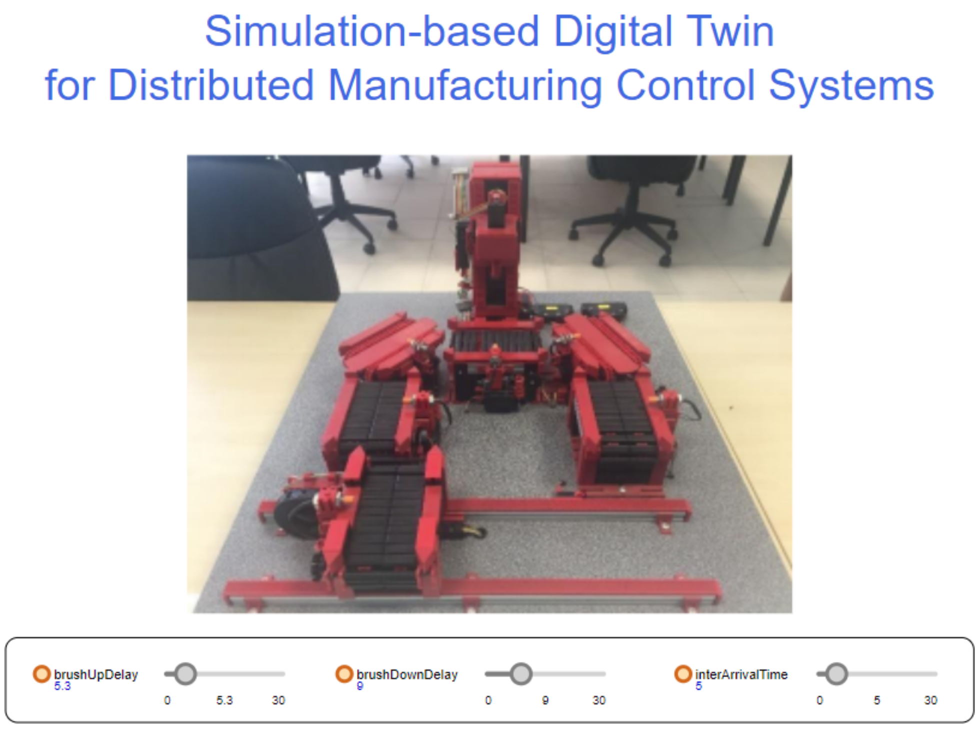
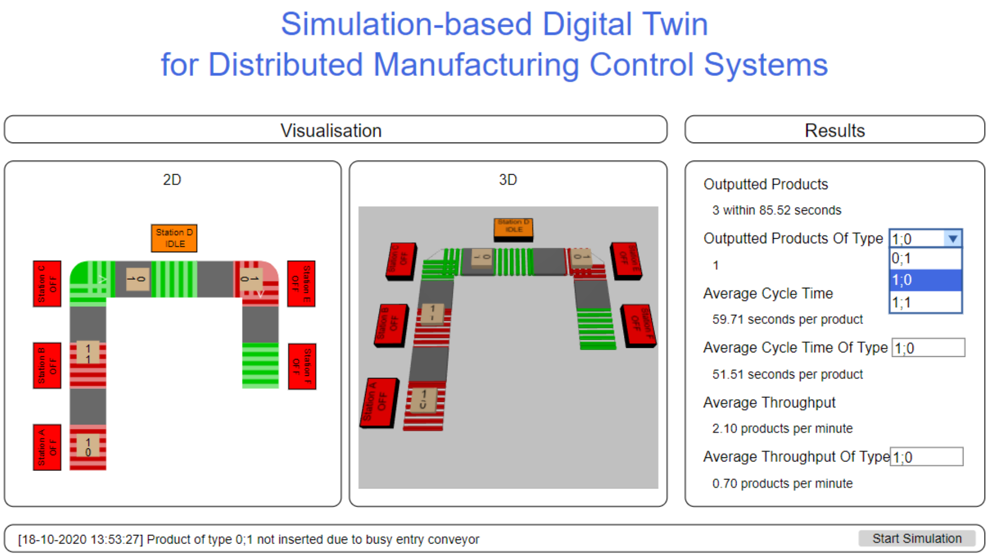

Simulation-based Digital Twin
for Distributed Manufacturing Control Systems

Figure 13. Simulation model's homepage.

As shown, it contains a picture of the physical system and three sliders that allow users to adjust the parameters described in the final paragraph of Section 2.1.

On the other hand, once the model is run, the main screen is presented, as shown in Figure 14. The latter includes both the 2D and 3D live views of the simulation model, a section for displaying global and type-focused predictions (depending on the product type selected on the respective combo box), an output section where feedback messages are printed and an interactive button used to start the simulation.


Simulation-based Digital Twin
for Distributed Manufacturing Control Systems
Visualisation
Results
2D
3D
Station D IDLE
Start Simulation

Outputted Products
3 within 85.52 seconds
Outputted Products Of Type 1;0
0;1
1;0
1;1
1
Average Cycle Time
59.71 seconds per product
Average Cycle Time Of Type 1;0
51.51 seconds per product
Average Throughput
2.10 products per minute
Average Throughput Of Type 1;0
0.70 products per minute

[18-10-2020 13:53:27] Product of type 0;1 not inserted due to busy entry conveyor

Figure 14. Simulation model's main screen.

Note that this button can only be pressed once, by which time the *loadState* method is invoked, every *hold* block is unlocked and the first attempt to insert a product is made. For this reason, the model must be reset before running a new simulation. Furthermore, a logbook is written during the simulation and saved when the application is stopped.

2.2.2. The Database File

According to the model's implementation, the database file must effectively describe the system's current state and the remaining production plan. Thus, deciding which information was needed in order to depict the system's state and the next products to be inserted was fundamental. After considering all of the system's relevant aspects, the following structure emerged.

The first six lines are mandatory and contain a maximum of five fields that describe each conveyors' initial state, sorted as follows.

- Timestamp—UNIX timestamp, in milliseconds, related to the corresponding conveyor's state last update.
- Station Flag—indicates whether a conveyor has a station attached to it (1) or not (0).
- Brush Ups—amount of BrushUp skills consumed by the product which is currently placed at a conveyor (−1 means that the conveyor is empty).
- Brush Downs—amount of BrushDown skills consumed by the product which is currently placed at a conveyor (−1 means that the conveyor is empty).
- Process Finished Flag—indicates whether the product placed at the respective conveyor has already been processed (0) or not (1). This field is only inserted when the corresponding conveyor is not empty and has a station plugged into it.

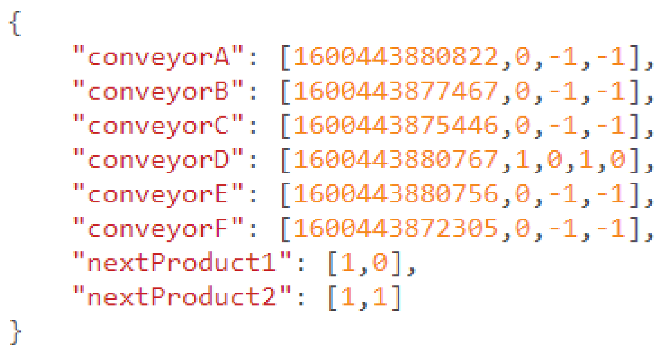
Each of the remaining lines is optional and has two mandatory fields: Brush Ups and Brush Downs, representing one product from the production plan, sorted by insertion priority.

In the example above, conveyor D has a station plugged into it and is carrying a processed product of type two. Two products, of types one and three, will be inserted next.

2.2.3. The API

The API's development was carried out using the Flask framework. Taking into account that its main purpose is reading or editing the *csv* file, the API only has two services.

On the one hand, a *POST* request may be sent through localhost:5000/api/v1/updateCSV, triggering the *updateCSV* method with the aim of modifying the file. The request's body must have a JavaScript Object Notation (JSON) message containing the fields from Section 2.2.2, as shown in Figure 15.



```
{
    "conveyorA": [1600443880822,0,-1,-1],
    "conveyorB": [1600443877467,0,-1,-1],
    "conveyorC": [1600443875446,0,-1,-1],
    "conveyorD": [1600443880767,1,0,1,0],
    "conveyorE": [1600443880756,0,-1,-1],
    "conveyorF": [1600443872305,0,-1,-1],
    "nextProduct1": [1,0],
    "nextProduct2": [1,1]
}
```

Figure 15. Sample JSON message.

This JSON sample, which maps each conveyor's initial state to its name and the product type to their order in the production plan, would generate the *csv* file from Listing 1. The *updateCSV* method behaves as portrayed by Figure 16.

Listing 1: Sample *csv* file.

```
1600443880822;0;-1;-1
1600443877467;0;-1;-1
1600443875446;0;-1;-1
1600443880767;1;0;1;0
1600443880756;0;-1;-1
1600443872305;0;-1;-1
1;0
1;1
```

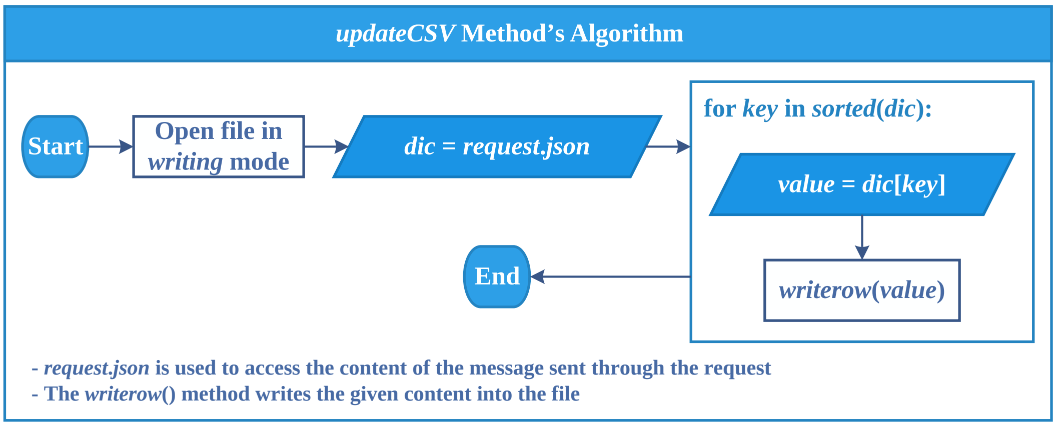

updateCSV Method's Algorithm
Start
Open file in *writing* mode
dic = *request.json*
for *key* in *sorted*(*dic*):
value = *dic*[*key*]
writerow(*value*)
End
- *request.json* is used to access the content of the message sent through the request
- The *writerow*() method writes the given content into the file

Figure 16. *updateCSV* method's algorithm.

After opening the file for writing, the content of the message is stored in a dictionary. Then, for each key, the corresponding value is written into the file.

On the other hand, a *GET* request can be sent through the localhost:5000/api/v1/getProducts Uniform Resource Locator (URL), which calls the *getProducts* method with the goal of obtaining the list of products to be added to the production line, i.e., the remaining production plan. In this case, no information needs to be sent in the request's body. The *getProducts* method's behaviour is illustrated in Figure 17.

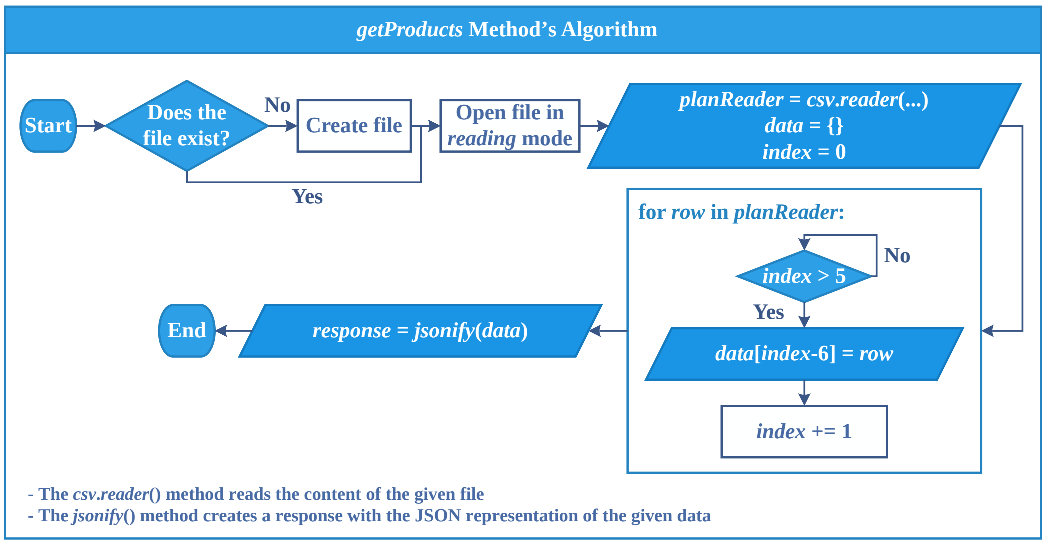

getProducts Method's Algorithm
Start
Does the file exist?
No
Yes
Create file
Open file in reading mode
planReader = csv.reader(...)
data = {}
index = 0
for row in planReader:
index > 5
No
Yes
data[index-6] = row
index += 1
End
response = jsonify(data)
- The csv.reader() method reads the content of the given file
- The jsonify() method creates a response with the JSON representation of the given data

Figure 17. *getProducts* method's algorithm.

Whenever the previous service is called, the file is created if it does not exist yet and, afterwards, it is opened for reading. Followingly, the file's content is iterated and, from its seventh line on, the information is stored in an auxiliary array. This is due to the fact that the first six lines are always reserved for the initial state information. Finally, the data are converted into the JSON notation and returned to the MAS.

2.2.4. The Multi-Agent System

The original version of the MAS was developed using Java Agent DEvelopment Framework (JADE) but no communications with external applications were allowed. Furthermore, products used to be manually added, which did not match the desired working mode for this project, where they ought to be inserted periodically. Consequently, in order to ensure the synchronisation between the manufacturing unit and the newly-created DT, a new Java class and a GUI were added to the initial project.

The *restClient* class contains auxiliary data structures used to store the system's current status, the remaining production plan and statistics data. Aside from that, it includes methods which interact with the aforementioned variables, keep the database file up to date, using the API, and compute the real KPIs. Furthermore, a *Ticker Behaviour*, from JADE, was used to ensure that an attempt to add a product to the network is made every five seconds.

In order to provide an intuitive way of describing the production plan to the MAS, a GUI with two sections was created. The first section, dedicated to the production pattern, is where the products' types are inserted and sorted into the desired order, whereas the second one is where the plan size is defined and the instruction to create the plan is given. This graphical interface is represented in Figure 18.

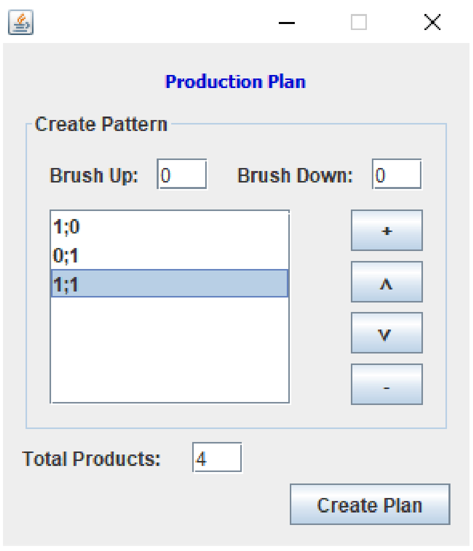


Figure 18. Production plan GUI.

In order to add a new product type, the amount of Brush Up and Brush Down skills has to be written in the respective text fields and the "+" button has to be pressed. In case the user mistakenly inserts a product type, it can be removed using the "−" button. Moreover, there are two buttons which move the selected product type up or down in the displayed pattern.

After describing the intended production pattern, the total amount of products shall be written into the respective text field and the "Create Plan" button must be pressed. If the content of the text field is numeric and the pattern does not contain any invalid product types, according to the ontology, the plan is created and the button is disabled so as to avoid a new attempted creation. In the example from the previous figure, the production plan would consist of a complete pattern, i.e., ProductType1 → ProductType2 → ProductType3, followed by a product of type one.

Once the plan has been defined, the MAS becomes autonomous and, inclusively, generates a *csv* file where the entry and exit timestamps of each processed product are stored. These can be used in case the user wants to calculate KPIs that are not automatically computed.

3. Demonstration and Validation

3.1. The Demonstrator

For validating the digital model, its predictions were compared with the values derived from the physical twin, whose components are shown in Figure 19.

For each simulation, the ensuing steps must be followed. After executing the Java project, a product has to be put on conveyor A. In fact, whenever this conveyor becomes empty, a new product must be added so that, when the next agent is launched, it starts moving immediately. Afterwards, the product travels from one conveyor to the following until the ResourceBrush's position is reached. Then, the due skills are executed, after which the product resumes its journey towards the end of the network. By the time it gets there, the agent is killed and the product must be manually removed.

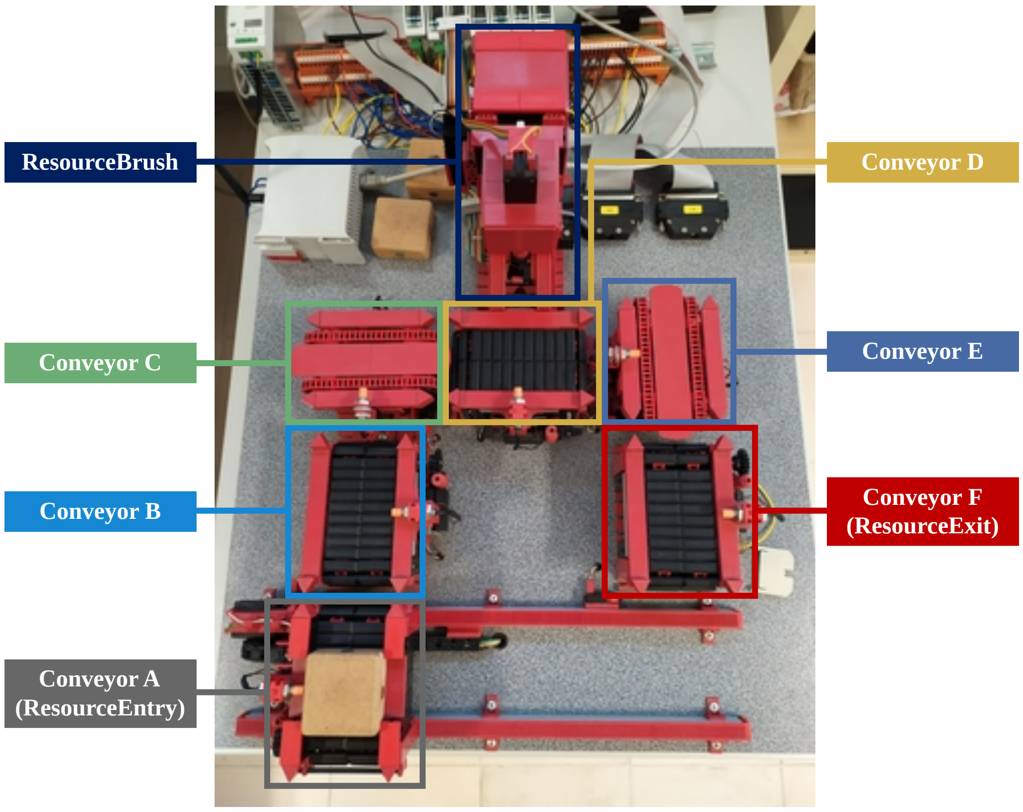


Figure 19. Manufacturing unit simulator's components.

In spite of the physical station being immovable, the ResourceBrush may be plugged into any conveyor. However senseless in a real-life context, this nuance does not affect the obtained results because the processing delays are independent of the resource's location.

3.2. Results Analysis

Before using the model to make predictions, each skill's processing time and the model's conveyors speed had to be tuned so that the simulation closely matched the physical system's behaviour. For this first calibration, single-product production plans were considered with a station at conveyor D. It was assumed that all the conveyors moved at the same speed, regardless of the station's position.

In order to assess this calibration's impact, twelve scenarios were tested on both systems (the simulation model and the physical demonstrator) for each station's possible location:

- Production pattern—six combinations of ProductType1, ProductType2 and ProductType3.
- Production plan size—ten or twenty products.

For each conducted test, the total processing time and the two KPIs below were extracted.

- Cycle time (CT)—average time spent by products on the line, in *s/product*.
- Throughput (Tp)—average amount of exported products per time unit, in *products/min*.

The first analysis unveiled some traits of the system's behaviour. The scenarios with a station at conveyor A presented the least accurate predictions, because products continuously communicate with the entry conveyor until they are allowed to enter the network, which makes the MAS become unexpectedly slower. As the station advances from one conveyor to the following, between B and D, the model gradually becomes less accurate. This is due to the fact that, the further the station is from the entry conveyor, the more products get stuck at the network, which leads to an increase in the amount of communications and, consequently, to greater delays. However, the change in the station's location from conveyor D to E is a turning point on this regard. From this conveyor on, the accumulation takes longer to occur and there is no such growth in the amount of communications, resulting in better predictions.

These discrepancies led to a second calibration. For the simplicity's sake, only two possible locations for the station (conveyors B and D) and the two scenarios below were tested:

- Production pattern—1→2→3 or 3→2→1.
- Production plan size—twenty products.

Besides, distinct speeds were chosen for each model's conveyor, through trial-and-error, taking into account the station's position as well.

Afterwards, all twelve scenarios were repeated with a station at conveyor B or D, the best and the second worst cases, respectively. As expected, the second calibration managed to improve the overall model's accuracy for both setups.

Nevertheless, the physical system still became quite unstable upon the products' accumulation, possibly due to the increase in the amount of agents and, consequently, communications. Moreover, in spite of being possible to adjust both the total time and the throughput for a specific production plan, it was achieved by reducing the conveyors' velocity, which spoiled each product's cycle time. For that reason, it is safe to assume that such change is insufficient.

Since modelling the delays transcended the scope of this work, a last verification was performed so as to ascertain whether processing a sufficiently large amount of products helped dissimulating such latency. For that, the scenarios from the second calibration were rerun with larger production plans, but the KPIs calculations still considered sets of twenty consecutive products.

The predictions' relative error for different production plan sizes, with a station at conveyor B, are represented in Figure 20.

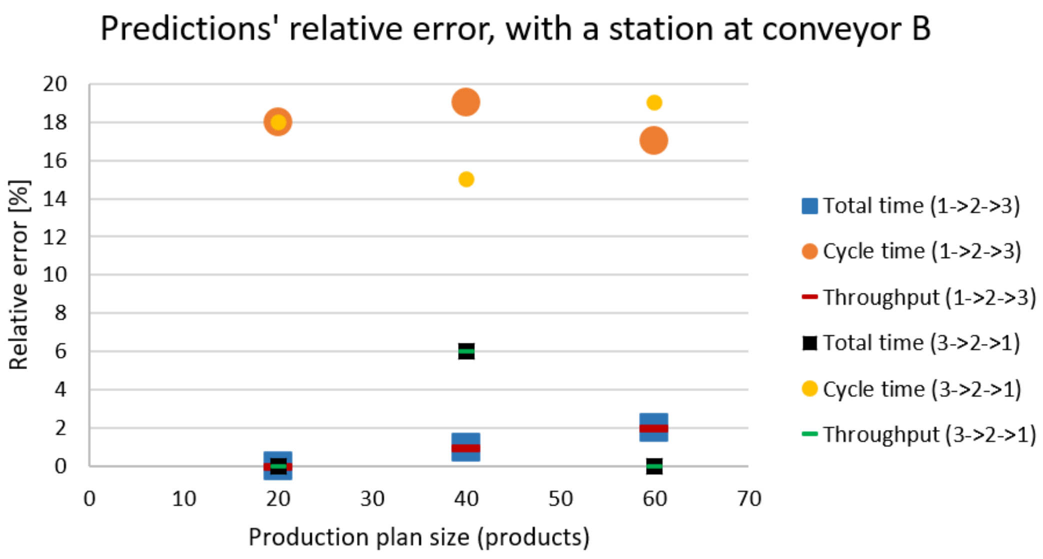


Figure 20. Predictions' relative error as a function of the plan size, with a station at conveyor B.

The corresponding values with a station at conveyor D are portrayed in Figure 21.

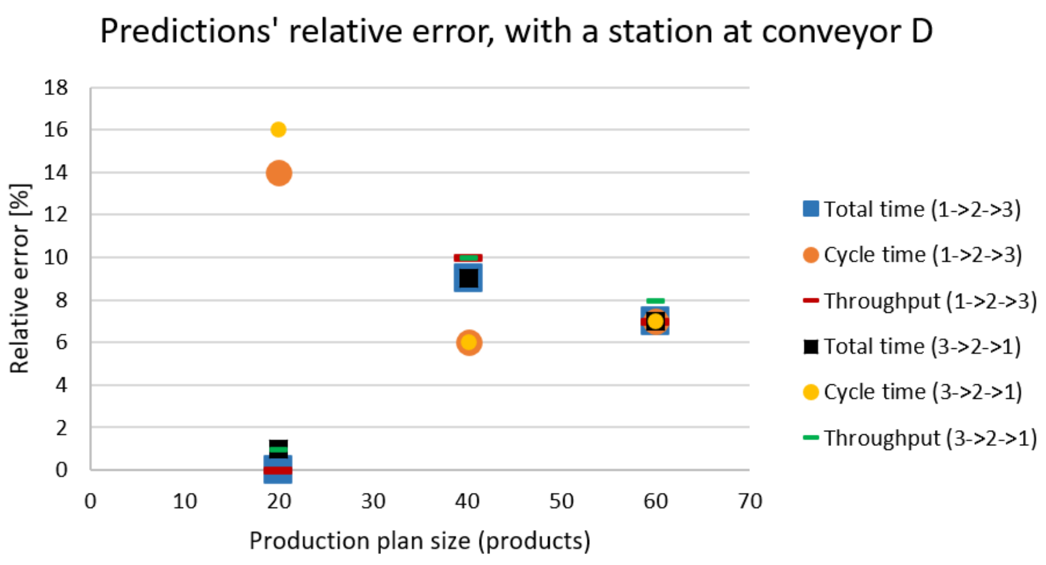


Figure 21. Predictions' relative error as a function of the plan size, with a station at conveyor D.

A closer look at the previous plots suggests that, regardless of the station's position, the relative error of each KPI's predictions did not directly increase with the production plan's dimension.

Finally, since they represented the same practical situation, the KPIs for the twenty intermediate products out of sixty were calculated and compared with the values obtained for the last twenty products out of forty in the corresponding test cases. The mismatches derived from this confrontation indicated that the manufacturing unit will often originate different results for the same test case. The most likely reason is the casual network instability, which may be exacerbated by the affluence of communications between agents depending on the amount of products being processed at a certain time. Such unsteady nature of the physical demonstrator partially explains the inaccuracies in the model's predictions.

3.3. Discussion

As presented at the beginning of the article, the application of distributed manufacturing control is quite difficult to predict and apply due to self-organized behaviour and the non-existence of a central point. Based on the study and results obtained, the authors believe that DTs' conjunction with these approaches can reduce the existent gap between the research approaches and real applications. The authors also understand that the results are promising but not mature enough to direct application. One of the reasons is that the existing simulation tools are not designed to deal with this system type. So it is necessary to develop new tools to increase the similarities of the control logic with the simulation model or improve the integration mechanisms to overcome these limitations.

Despite the limitations found during the design and developments, it is possible to verify the proposed three-layered architecture's potential to interface a simulation-based DT on top of an existing or new distributed manufacturing control system. The proposed architecture with the different modules proposed and described in Section 2 can be used as a baseline for future developments. To summarise, the utilisation of an API to regularly store in a database the current state of the real system can be used as the starting point for a simulation environment. In conjunction with the state stored in the database, an accurate simulation model can approximately mimic the system's behaviour at a much higher speed, allowing to predict the real system's performance.

Despite the existent limitations, the proposed research contributes to the utilisation of DTs to allow the creation of a more harmonised ecosystem with self-organised manufacturing control and human personnel. In this context, DTs allow the human to understand the self-organised systems' trends. This feature reduces the impact of unpredictability in organisations, suppliers and costumers.

Although the proposed DT approach can help humans using these systems daily, humans are also critical and fundamental for developing and maintaining these simulation-based DTs due to their expertise. The creation of a better and more accurate simulation depends not only on the simulation environment but also on the amount of knowledge available to be applied during the model's creation. A more accurate model will lead to creating a more similar to the real environment model, increasing its utility.

4. Conclusions and Future Work

The usage of DTs is still at an early stage. As a result, standards for implementing them are lacking. With the aim of covering this major gap, a general architecture was pondered jointly with the digital model's implementation, so that future solutions may be based on it.

The suggested architecture focuses on dividing each component of a CPS. Moreover, none of the used technologies narrows down the range of target-problems, given that widely-compatible programming languages and file formats were opted for. This way, it is believed that a considerable percentage of the oncoming solutions will be safe to rely on an architecture along the lines of the one hereby introduced.

It is possible to conclude two main ideas from the results of the proposed article. The first one is that the utilisation of simulation-based DTs to predict the behaviour and, more specifically, a distributed manufacturing control system's performance can be useful. Due to the similarity of the values predicted and collected from the real system, the results are promising. A prediction, even not very accurate, can constitute a relevant contribution to applying these distributed control approaches because it is possible to approximately predict the system's performance and the self-organised ecosystem's behaviour, which is impossible to do presently. The second lesson learnt from this research is that the utilisation of simulation to mimic a distributed control logic needs to mimic not only the logic behind each decision and interaction, but also other characteristics like network connections or the amount of cyber-physical modules, and possibly the number of computational devices available and their performance.

To sum up, the discovered research gaps were not fully covered, because the network's stability and the amount of products being simultaneously processed substantially influenced the model's performance. However, these breakthroughs may be cornerstones of further research on the topic, provided that such drawbacks are taken into consideration on future DTs' implementations.

As future work, since the communication delays were found to be the most likely cause of inaccuracy, the idea of modelling them arose. Moreover, given that the manufacturing unit can deal with changes in the resource's location halfway through the simulation, it would be valuable to provide the DT with such capability. Finally, it is advisable that the developed model be exported as a stand-alone app and launched from the MAS project. Thereafter, the DT becomes attached to its physical counterpart's implementation.

Author Contributions: Conceptualization, A.D.R. and J.B.; methodology, G.R.R. and A.D.R.; software, G.R.R. and J.T.; validation, G.R.R., A.D.R. and J.T.; formal analysis, G.R.R. and A.D.R.; investigation, G.R.R. and A.D.R.; writing—original draft preparation, G.R.R. and A.D.R.; writing—review and editing, G.R.R. and A.D.R.; supervision, A.D.R. and J.B.; funding acquisition, A.D.R. and J.B. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the FCT/MCTES project CESME—Collaborative & Evolvable Smart Manufacturing Ecosystem, funding PTDC/EEI-AUT/32410/2017.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AI	Artificial Intelligence
API	Application Programming Interface
AR	Augmented Reality
CTS	Centre of Technology and Systems
CPS	Cyber-Physical System
CPPS	Cyber-Physical Production System
DCL	Distributed Control Layer
DT	Digital Twin
GUI	Graphical User Interface
I4.0	Industry 4.0
IL	Integration Layer
IoT	Internet of Things
JADE	Java Agent DEvelopment Framework
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
KRI	Key Result Indicator
MAS	Multi-Agent System
NSF	National Science Foundation
OPC-UA	Open Platform Communications Unified Architecture
OWL	Ontology Web Language
PI	Performance Indicator
PLC	Programmable Logic Controller
PLM	Product Lifecycle Management
RICS	Robotics & Industrial Complex Systems
SL	Simulation Layer
URL	Uniform Resource Locator

References

1. Büchi, G.; Cugno, M.; Castagnoli, R. Smart factory performance and Industry 4.0. *Technol. Forecast. Soc. Chang.* **2020**, *150*, 119790. [CrossRef]
2. Raj, A.; Dwivedi, G.; Sharma, A.; Beatriz, A.; Sousa, L.D. International Journal of Production Economics Barriers to the adoption of industry 4.0 technologies in the manufacturing sector: An inter-country comparative perspective. *Int. J. Prod. Econ.* **2019**, 107546. [CrossRef]
3. Li, G.; Hou, Y.; Wu, A. Fourth Industrial Revolution: Technological drivers, impacts and coping methods. *Chin. Geograph. Sci.* **2017**, *27*, 626–637. [CrossRef]
4. Negri, E.; Fumagalli, L.; Macchi, M. A review of the roles of digital twin in cps-based production systems. *Procedia Manuf.* **2017**, *11*, 939–948. [CrossRef]
5. Lu, Y.; Xu, X. Resource virtualization: A core technology for developing cyber-physical production systems. *J. Manuf. Syst.* **2018**, *47*, 128–140. [CrossRef]
6. Zeng, J.; Yang, L.T.; Lin, M.; Ning, H.; Ma, J. A survey: Cyber-physical-social systems and their system-level design methodology. *Future Gener. Comput. Syst.* **2016**, *105*, 1028–1042 [CrossRef]
7. Alguliyev, R.; Imamverdiyev, Y.; Sukhostat, L. Cyber-physical systems and their security issues. *Comput. Ind.* **2018**, *100*, 212–223. [CrossRef]
8. Monostori, L. Cyber-physical production systems: Roots, expectations and R&D challenges. *Procedia CIRP* **2014**, *17*, 9–13.
9. Hehenberger, P.; Vogel-Heuser, B.; Bradley, D.; Eynard, B.; Tomiyama, T.; Achiche, S. Design, modelling, simulation and integration of cyber physical systems: Methods and applications. *Comput. Ind.* **2016**, *82*, 273–289. [CrossRef]
10. Dias-Ferreira, J.; Ribeiro, L.; Akillioglu, H.; Neves, P.; Onori, M. BIOSOARM: A bio-inspired self-organising architecture for manufacturing cyber-physical shopfloors. *J. Intell. Manuf.* **2016**, 1–24. [CrossRef]
11. Ribeiro, L.; Rocha, A.; Veiga, A.; Barata, J. Collaborative routing of products using a self-organizing mechatronic agent framework—A simulation study. *Comput. Ind.* **2015**, *68*, 27–39. [CrossRef]
12. Leitão, P.; Karnouskos, S. *Industrial Agents: Emerging Applications of Software Agents in Industry*; Morgan Kaufmann: San Francisco, CA, USA, 2015.
13. Leitão, P. Agent-based distributed manufacturing control: A state-of-the-art survey. *Eng. Appl. Artif. Intell.* **2009**, *22*, 979–991. [CrossRef]

14. Rocha, M.P.D.S.V.P. Risk of Employing an Evolvable Production System. Master's Thesis, Universidade Nova de Lisboa, Lisboa, Portugal, 2015.
15. Grieves, M. *Digital Twin: Manufacturing Excellence through Virtual Factory Replication*; White Paper; Florida Institute of Technology: Melbourne, FL, USA, 2014; Volume 1; pp. 1–7.
16. Kritzinger, W.; Karner, M.; Traar, G.; Henjes, J.; Sihn, W. Digital Twin in manufacturing: A categorical literature review and classification. *IFAC-PapersOnLine* **2018**, *51*, 1016–1022. [CrossRef]
17. Samir, K.; Khabbazi, M.R.; Maffei, A.; Onori, M.A. Key Performance Indicators in Cyber-Physical Production Systems. *Procedia CIRP* **2018**, *72*, 498–502. [CrossRef]
18. Qi, Q.; Tao, F.; Zuo, Y.; Zhao, D. Digital twin service towards smart manufacturing. *Procedia CIRP* **2018**, *72*, 237–242. [CrossRef]
19. Zhang, C.; Xu, W.; Liu, J.; Liu, Z.; Zhou, Z.; Pham, D.T. A reconfigurable modeling approach for digital twin-based manufacturing system. *Procedia CIRP* **2019**, *83*, 118–125. [CrossRef]
20. Haag, S.; Anderl, R. Automated Generation of as-manufactured geometric Representations for Digital Twins using STEP. *Procedia CIRP* **2019**, *84*, 1082–1087. [CrossRef]
21. Cimino, C.; Negri, E.; Fumagalli, L. Review of digital twin applications in manufacturing. *Comput. Ind.* **2019**, *113*, 103–130. [CrossRef]
22. Ait-Alla, A.; Kreutz, M.; Rippel, D.; Lütjen, M.; Freitag, M. Simulation-based Analysis of the Interaction of a Physical and a Digital Twin in a Cyber-Physical Production System. *IFAC-PapersOnLine* **2019**, *13*, 1331–1336. [CrossRef]
23. Lu, Y.; Liu, C.; Wang, K.I.; Huang, H.; Xu, X. Digital Twin-driven smart manufacturing: Connotation, reference model, applications and research issues. *Robot. Comput.-Integr. Manuf.* **2020**, *61*, 101837. [CrossRef]
24. Rocha, A.D.; Tripa, J.; Alemao, D.; Peres, R.S.; Barata, J. Agent-based plug and produce cyber-physical production system—Test case. In Proceedings of the IEEE International Conference on Industrial Informatics (INDIN), Helsinki, Finland, 22–25 July 2019; pp. 1545–1551. [CrossRef]
25. Zheng, L.; Xiao, J.; Hou, F.; Feng, W.; Li, N. Cycle time reduction in assembly and test manufacturing factories: A KPI driven methodology. In Proceedings of the 2008 IEEE International Conference on Industrial Engineering and Engineering Management, Singapore, 8–11 December 2008; pp. 1234–1238. [CrossRef]
26. Johnson, D.J. A framework for reducing manufacturing throughput time. *J. Manuf. Syst.* **2003**, *22*, 283–298. [CrossRef]